Informationstheorie (Oldenbourg, 1960)) although the letter H has no relation to the word entropy—a clear sign of Shannon's respect for Boltzmann's share in information theory.

On page 180: Ortvay proposed an axiomatic method for the systems theory: the same wrong line that McCulloch and Pitts pursued with their paper. Organic structures—languages included—do not have axiomatic nature; a systems theory cannot be built on such sharp logic (fuzzy logic is not much better). Logic and mathematical models must be included, but their interconnection has to be open or loose if the true idea of organization is to be modelled.

On page 186: A.D. Booth was not only a cristallographer; returned to England, he started to develop computers for exactly this purpose—a relationship that would deserve deeper investigation.

On page 187: Heinz von Foerster was charged by W.S. Culloch to edit the printed second half of the symposia (6 . . . 10). Von Foerster followed A. Samuel in the chair for electronics at the University of Illinois in Urbana and in a certain way he continued John von Neumann's work on the brain and the computer.

On page 200: Multiplexing here is neither time nor frequency multiplexing, but space multiplexing, an unusual application of the word multiplexing.

In summary: John von Neumann *is* a hero of computing—he does not need the majority element. His weight and his influence, however, have distorted a little the historic accounts. The public opinion majority elements have reduced the shares of other contributors in them and have increased the (already big) share of John von Neumann. A chapter on the weaknesses and on the negative influence of the hero would be of no less importance.

And the European reader waits for a collection of John von Neumann anecdotes. One can hear more than one in the US, but for some reason Americans do not cultivate anecdotes (except as footnotes like the section in the Annals). A genius like John von Neumann, I dare say, is difficult to present by his scientific achievements which extend beyond the horizon not only of the average reader. He could get a much more distinctive profile by a baker's dozen of anecdotes whose pointwise flashes would produce a three-dimensional picture of the extraordinary human being John von Neumann. A second volume by William Aspray?

Heinz Zemanek
Vienna, Austria

L.C. Paulson, *ML for the Working Programmer* (Cambridge University Press, Cambridge, England, 1991), Price £27.50, $49.50 (hardcover), ISBN 0-521-39022-2.

Based on his experience with teaching, the author has written a book for an audience which he omits to identify clearly. The title indicates an audience of working programmers, whoever that may be. One objective is to introduce functional programming techniques based on ML. Another is to present ML rather extensively as a programming language to people who haven't been exposed to that kind of languages before.

It is definitely a delightful book for the reviewer. Written in a very direct style and organized differently from many other books on functional programming. Besides the bulk of text contained in ten chapters the book contains a Preface, a Bibliography, Standard ML Syntax Charts, an Index, and a list of Predefined Identifiers: 430 pages altogether.

The first four chapters present the basic language primitives and illustrate programming with a functional subset of ML and without higher-order functions. Some of the problems used in the illustrations are refreshingly traditional: computation of square roots, matrix multiplication, Gaussian elimination, topological sorting, random number generation, and various sorting algorithms. The presentation of language primitives is thorough, but marred in a few places by simple blunders (e.g. "If the divisor is non-zero [...] an error will be signalled" (page 55), and "The collection of values denoted by a type scheme is essentially the intersection of all its instances" (page 57)).

Chapters 5 through 8 treat more advanced issues of ML: higher-order functions, proof principles, modules, references. Linguistic aspects are put into focus and the examples are more in line with other presentations of functional programming. Examples include lazy evaluation techniques, strategies for searching in trees, arrays, priority queues, and cyclic data structures. The author explains in depth such concepts as formal proofs, specifications, sharing among modules, assignments and weak polymorphism. This is definitely one of the book's strong points, but it is tough reading. It may be a problem that modules are treated more from an implementer's view than from a user's.

Each of the final two chapters is devoted to a large example: writing interpreters for the lambda calculus, and a tactical theorem prover. It is hard to tell whether the working programmer will find these relevant or interesting. The university professor probably does, and the reviewer does so for sure.

In summary I find this an admirable book. Its coverage is very wide, and I would like to use the first chapters for an early course in a computing science curriculum. I have no doubt that the first four chapters can be studied by a working programmer without assistance from an instructor. The remaining part of the book will fit nicely into one or two university courses later on. I doubt whether it is suitable for self-study except for a rather narrow audience.

Jørgen Steensgaard-Madsen
Technical University of Denmark
Lingby, Denmark